BEST AVAILABLE COPY

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G07F 7/10	A1	(11) International Publication Number: WO 99/38131
		(43) International Publication Date: 29 July 1999 (29.07.99)

(21) International Application Number: PCT/GB99/00209

(22) International Filing Date: 21 January 1999 (21.01.99)

(30) Priority Data:
60/072,561 22 January 1998 (22.01.98) US

(71) Applicant: MONDEX INTERNATIONAL LIMITED [GB/GB]; 1st floor, 47–53 Cannon Street, London EC4M 5SQ (GB).

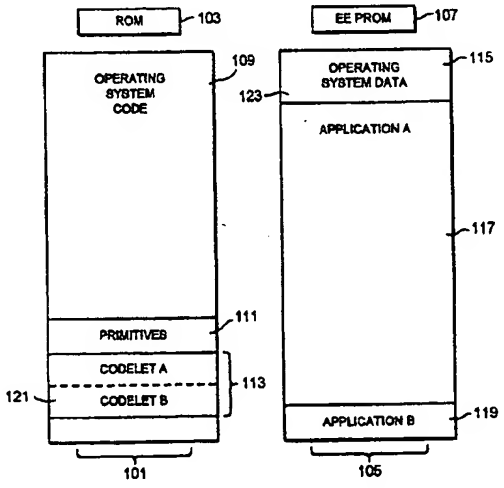
(72) Inventor: PEACHAM, David; 4 Lynwood, Groombridge, Tunbridge Wells, Kent TN3 9LX (GB).

(74) Agent: REDDIE & GROSE; 16 Theobalds Road, London WC1X 8PL (GB).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: CODELETS



(57) Abstract

A system and method for efficiently storing programming instructions in a microprocessor based system where codelets which includes program instructions written in a non–native programming language (such as MEL or C) are stored in a read only portion of memory. The location of the codelets are stored in an address table which is accessed by an operating system when an application calls the codelets during execution. At that time, the microprocessor accesses the codelet instructions until the codelet function is complete. By storing codelets in read only memory which is cheaper and takes up much less physical space than alterable memory (such as EEPROM), more programming instructions can be stored in the same amount of physical space. Additionally, since codelets are written in non–native programming languages, they become platform independent since they can be compiled by different compilers to run on any platform.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

CODELETS

SPECIFICATION

PRIORITY APPLICATION

This application claims priority to United States Provisional application 60/072,561 filed on January 22, 1998, and entitled "CODELETS", which is hereby incorporated by reference.

RELATED APPLICATION

This application is related to United States application No. 09/064,915 filed on April 23, 1998, entitled "MULTI-APPLICATION IC CARD WITH DELEGATE FEATURE" and assigned to Mondex International Limited, which is hereby incorporated as Annex A.

FIELD OF INVENTION

This application relates to a system and method for improving the storage capacity and efficiency of memory management, in particular in an integrated circuit card, through the selective storage of programming instructions.

BACKGROUND OF INVENTION

Integrated circuit (IC) cards are becoming increasingly used for many different purposes in the world today. An IC card typically is the size of a conventional credit card which contains a computer chip including a microprocessor, read-only-memory (ROM), electrically erasable programmable read-only-memory (EEPROM), an Input/Output (I/O) mechanism and other circuitry to support the microprocessor in its operations. An IC card can be application specific or may contain multiple applications in memory. MULTOS™ is a multiple application operating system which runs on IC cards, among other platforms, and allows multiple applications to be executed on the card itself. This allows a card user to run many programs stored in the card (for example, credit/debit, electronic money/purse and/or loyalty applications) irrespective of the type of terminal (i.e., ATM and/or POS) in which the card is inserted for use.

IC cards typically have limited storage capacity due to the size and cost restraints of locating memory on the card. Multi-application smart cards have their applications written in a programming language that are typically stored in the EEPROM

whose contents can be changed during the lifetime of the card. One example of a programming language used in IC cards is Multos Executable Language (MEL). The MEL program instructions are read from EEPROM, an alterable memory, when they are executed and are interpreted by the operating system stored in ROM.

The ROM on the IC card includes the operating system written in assembly language code for the particular integrated circuit configuration (native language type code). The operating code stored in ROM is fixed when the ROM is initially written and the information stored in ROM will not change for the life of the card.

One of the concerns with multi-application IC cards is that when attempting to store multiple applications on the card, the memory constraints of the EEPROM becomes significant. Currently, the size of a typical EEPROM on an IC card is 8K bytes. The size of an application program may be 3.3K for an electronic money (purse) application. Moreover, the application typically has data associated with the code, which could be about 3K bytes of memory for an electronic money application. Therefore, the application including the code and the data typically requires 6.6K of memory space. In addition, the operating system requires data to be stored in EEPROM, which data is used during the operation of the operating system. This overhead requirement is generally about 1K of space in EEPROM. Consequently, if a purse application is stored in EEPROM, 7.6K of memory will be needed, leaving only .4K of

EEPROM available for a second application. This is not an acceptable option for an efficient and effective multi-application IC card system.

With regard to memory space in ROM, a typical multiple application operating system code requires 17.5K of the 24K of available memory in ROM. Therefore, 6.5K of ROM is unused and can never be used after the card is distributed to the public because ROM can only be configured once. Moreover, ROM memory is approximately six times more dense than EEPROM memory, meaning that 1K of EEPROM takes up six times more room on the chip than 1K of ROM. As a result, it would be advantageous to (1) fully utilize any unused memory space in ROM and (2) use as much ROM as possible instead of EEPROM to minimize the size of memory on the integrated circuit in the card.

SUMMARY OF THE INVENTION

The invention is directed to a system and method of efficiently storing programming instructions in a microprocessor based system with memory size and/or cost constraints, such as an IC card. Codelets include programming instructions written in non-native code and are stored in a read-only portion of memory. The memory address of the codelet is stored in an address table which the operating system for the microprocessor based system accesses in order to execute the codelet. An application residing in the alterable portion of memory will call the codelet to be executed with a

program instruction, and the operating system will look up the memory address of the codelet from the address table and execute the codelet's program instructions to perform its designated function. By storing the codelet in read-only memory, the programming instructions are stored in a memory which is cheaper and physically smaller than alterable memory so more overall programming instructions can be stored in the overall memory system. Additionally any extra space in the read-only memory can be fully utilized by using the codelets.

The codelet is written in a non-native programming language such as MEL or C so that the instructions will be interpreted by the operating system in order to execute them. This allows different platforms to use the same codelet because each operating system will convert the non-native programming instructions to machine readable instructions for the particular microprocessor. Thus the codelets can be used with different types of platforms without the need to translate the instructions for specific microprocessors. When the codelets are executed, they will act upon the data used by the application which accessed the codelet in a preferred embodiment.

BRIEF DESCRIPTION OF THE DRAWINGS

Further objects, features and advantages of the invention will become apparent from the following detailed description taken in conjunction with the accompanying figures showing illustrative embodiments of the invention, in which

Figure 1 is a diagram of the memory of an IC card configured in accordance with an embodiment of the invention;

Figure 2 is a flowchart of the steps for performing a codelet query; and

Figure 3 is a block diagram of an IC card chip which can be used in accordance with an embodiment of the invention.

Throughout the figures, the same reference numerals and characters, unless otherwise stated, are used to denote like features, elements, components or portions of the illustrated embodiments. Moreover, while the subject invention will now be described in detail with reference to the figures, it is done so in connection with, and by way of example of, the illustrative embodiments. It is intended that changes and modifications can be made to the described embodiments without departing from the true scope and spirit of the subject invention as defined by the appended claims.

DETAILED DESCRIPTION OF THE INVENTION

Greater and fuller utilization of ROM and efficient overall use of memory is accomplished through the use of "codelets," which are sets of instructions written in a programming language (not native language code). These codelets can be stored in ROM so as to maximize the usage of memory and allow ROM to store complete applications as well as primitives. The codelet can be as small as one instruction or as large as will fit into the remaining ROM memory space. For example,

the purse application described in the background can be stored in ROM when the card is initialized in order to free up space in EEPROM, the alterable memory, for additional applications which can be loaded at any time.

The codelet is assigned a name and that name is placed in an address table stored in EEPROM. When a codelet is called by another application, the address table gives the location of the first instruction in the codelet stored in ROM and each instruction is executed just as if the application were stored in EEPROM.

Also present in ROM can be subroutines called primitives written in a native language code for the microprocessor which can be called by either the operating system itself or by applications when they are executed. Primitives are written in the native language (i.e. assembler language) so that they can be executed very quicky and minimal interpretation of the instructions is necessary for execution. These primitives are collections of instructions which typically perform a desired function, such as a mathematical function. The instructions are never changed during the lifetime of the card. Any data used or accessed by the primitives are stored in EEPROM so that the contents of the data elements can change as necessary. Primitives are different than codelets because they are written in the native language code. Codelets allow programmers to use an easier and more programmer friendly programming language such as MEL or "C." Programming languages also allow programmers to do advanced features which may be hard to directly implement in an assembler native language.

Figure 1 shows an example of the memory configuration of ROM 101 (indicated by label 103) and of EEPROM 105 (indicated by label 107) which are located on an IC card. ROM 101 includes operating system code 109 stored in native language code (e.g., assembly language) which is run by the microprocessor to operate any application and perform card functions. Also stored in ROM are primitives 111 which are sets of instructions stored in the native language code on ROM and which are called by the operating system and/or an application being run on the card. Codelets 113 are sets of instructions stored in a programming language form (e.g., in MEL) in ROM which are called by another application or codelet. The operating system will not call codelets directly as part of the operating code because of the high speed required by the operating system. Program instruction sets called by the operating system are preferably primitives written in the native language codes.

Operating system data 115 is stored in EEPROM 105. This data is necessary for the operation of the operating system and is unavailable for applications to utilize. Application A space 117 comprising program code and data is entirely stored on EEPROM 105 and takes up most of the available space in the memory. Application B space 119 has very few lines of code and calls the codelet B 121 which is stored in ROM 101. The data associated with application B is stored in the EEPROM because the data may change. This configuration allows multiple applications to reside in EEPROM memory 105 by storing large blocks of code in ROM and calling the codelet instructions.

A codelet address table 123 resides in EEPROM 105 as part of the operating system data. This allows the operating system to locate the codelet when an application calls the codelet. Alternatively, the codelet address table 123 could reside in ROM 101 for addresses known when the card is first manufactured.

While EEPROM 105 is described as a preferred embodiment of an alterable memory, any other memory whose contents can be changed can also be used as an alterable memory.

As memory storage capabilities of EEPROM get larger with time, memory management will continue to be an important concern because more and more applications will be available on the card. Thus, a user of the card may choose different combinations of purse applications, credit/debit applications, loyalty programs such as frequent flyer reward programs, health information programs or catalog ordering programs. Each application will require a finite amount of space on the card and codelets will help utilize available memory space efficiently.

A codelet can be stored in EEPROM as well as in ROM, or any other memory area which can be addressed by the card. This includes additional EEPROMs if present or even external memory units which are accessible by the card. Codelets could also contain discrete portions of programs so that part of a program will reside in EEPROM and part of a program as a codelet on ROM. The codelet can be called by one

or more applications which are executed. A codelet is written in the application language (e.g. MEL) and is considered part of the program no matter where it resides.

Applications written in an application language will preferably run on a number of different platforms (i.e. different integrated circuits made by different manufacturers or different integrated circuit models made by the same manufacturer). It is desirable to have an application written once and allow the particular IC card to compile the application code to run the code. A primitive is written in native language code which is specific to the platform it resides on. A primitive for one platform will not run on the platform of a different manufacturer. Codelets transcend the limitations of a specific platform by being written in a program language which can be stored on any platform without lengthy pre-processing or without having to rewrite the code to fit the platform.

Codelets applicability are not restricted simply to IC cards, but can also be used in other microprocessor systems with memory constraints such as a computer watch or other item containing a processor.

Figure 2 is a flow chart of the steps for performing a query codelet. If an application needs to determine that a codelet has been stored on the IC card before it is called, it can execute a query codelet function. An exception to the general rule that codelets are written in an application programming language is the query codelet function, which is a series of program instructions that can be stored as a primitive. The query codelet function will check the address table stored in the memory of the IC card for the name of the codelet ————

which is being checked. If the codelet name has been stored, then a flag indicator such as the zero (Z) flag of a conventional Condition Code Register (CCR) is set to one (indicating a positive response) and the application can then successfully call the codelet which has just been checked. If the name of the codelet is not present in the address table, then a zero will be placed in the Z flag (indicating a negative response) and the application will not call the codelet to avoid an execution error and/or an abnormal end to the execution of the application.

Step 201 sets the variable codelet_id equal to the name of the requested codelet to be checked. The name can be any combination of letters, numbers or characters which identify the codelet. If the requested codelet name is zero, a predetermined wild card, then a match to a codelet name in the address table is always made and the Z flag is always set to one (a positive response). Step 203 then retrieves the address table from the memory of the IC card and in particular the portion containing the addresses of any stored codelets.

Step 205 sets the CCR Z flag to zero. The CCR register is preferably used as a mechanism to relay the answer to the codelet query to the application or other program instructions requesting the query. The CCR register is used because it is very fast and enables the requesting application to determine the outcome of the request without transferring data or looking up further variables in designated memory areas. The default response to a codelet query is negative indicating that the codelet has not

been stored on the IC card. The negative response is indicated by a zero value in the Z bit of the CCR register.

Step 207 checks if the name stored in codelet_id is present in the codelet address table, and if it is the Z flag in the CCR register is set to a positive state. A comparison is made of the codelet_id variable to the names of codelets in the address tables. If a match does not occur, the Z flag remains set to zero in the negative state. If a match occurs, then the codelet has already been stored and its name properly placed in the address table so the Z flag is set to a positive state, i.e. one. If codelet_ id is set to a value of zero, then an automatic match is recorded. This wild card feature can be used as needed by the programmer.

Step 209 then returns control of the processor to the application or other instruction set which inquired about the status of the codelet. The application can then successfully pass control of the microprocessor to the codelet instructions and have the codelet instructions executed if the codelet was found in the address table. If the codelet name was not found, the application can execute alternative instructions based on the negative result of the codelet check. The application developer can program for both alternatives.

Additional operational primitives besides query_ codelet can also be used in connection with codelets. A call_codelet primitive can be used to pass control over from the application to the codelet called in order to execute the codelet instructions. The

codelet to be executed is identified by the specific codelet ID. If the request codelet ID is set to zero a predetermined special value, then a specific known codelet address can be used to identify and execute the codelet outside of the address table. If the codelet_id is not zero, then the address for the codelet is looked up on the address table and that address is used to locate the start of the program instructions for the codelet. If the codelet ID is non-zero but does not appear in the address table, then an abnormal end occurs. However, this error can be avoided by performing a query_codelet function as described above.

The call_codelet function can be used, for example, to (1) pass control from an application to a codelet; (2) pass control from one codelet to another; or (3) pass control from a codelet to itself (which may be done for program memory management or other iterative programming reasons). The codelet acts on the data segment of the calling application and preferably has no data of its own. This ensures that there is no possibility of data leakage between applications and keeps the data secure and uncorrupted. The codelets utilize the program and data stacks and the application abstract architecture machine as described in United States application No. 09/064,915 entitled "Multi-Application IC Card With Delegate Feature" which has been incorporated by reference.

An efficient architecture for processing multi applications in an IC card is termed an Application Abstract Machine (AAM) architecture. The AAM Architecture applies to any platform independent of its hardware and enables developers to write

applications to store on the IC cards which are portable across many different types of platforms (e.g., IC cards built by different manufacturers with different processor configurations) without the need for knowledge about the specific hardware of the platform.

An application abstract machine (AAM), a term for the memory allocation and organization for the data stored and used by each application, is created for each application stored on the IC card which is executed by the processor on the card. In order to ensure data integrity and security when data is transferred between applications which are executed on the IC card, only one application on the IC card is allowed to be executed at a time. Each application has a data memory space which is virtually allocated and mapped onto the physical memory addresses available in the IC card memories. Data is then passed between two or more applications within a specified memory location and in a manner consistent with transferring data to an external terminal or device with which the IC card is securely interacting. At a general level, each AAM space created for each application being executed includes two separate address spaces, one for the program code itself and one for the program data which is stored and/or used by the application. The program data address space is effectively divided into three segments: a Static segment, a Dynamic segment and a Public segment. The Static, Dynamic and Public segments are logically mapped to the physical memory; they are virtual memory segments as opposed to physical memory segments. The AAM data address space is

preferably addressed and processed using seven different address registers and two control registers.

In a multi-application operating system, the operating system can configure each application to have its own logical address space. Each application therefore operates independently and if one application calls another application, the communication will follow procedures similar to talking to a terminal. This type of configuration ensures the integrity of the logical address space and enhances security of data flow between applications. Codelets, which can contain substantially all, or possibly all, of the instruction lines of the code, are written in the application language. By calling a codelet, the logical space for an application is maintained and the application instructions are simply located in a more efficient memory location. Moreover, because the I/O like parameters which are required between applications are not invoked when the codelet is called, codelets allow extra flexibility and efficiency in the operation of the IC card.

The pointers to the AAM data segments can be redirected when a codelet is called. Thus, where a codelet is stored in EEPROM, the static segment of the memory of the AAM will be made up of the codelet instructions while the dynamic segment of memory will still be that of the original application calling the codelet. This allows the codelet to perform its function on the applicable data and then return control to the application when it is complete. Because the codelets are preferably stored in a read-

only type memory, codelets typically do not have their own variable data but instead use the data from the application in a secure manner.

When control of the IC card processing switches from an application or other codelet to the codelet called, it is important that the operating system know the correct address of the codelet program instructions. Although the address of the codelet is stored in the address table, an additional check can be performed by setting an upper boundary so that the fetched instructions of the codelet by the operating system will be from a valid memory location. This primitive is called set_code_boundaries.

Figure 3 shows an example of a block diagram of an integrated circuit 380 located on an IC card chip which can be used in conjunction with the invention. The integrated circuit chip is located on a chip on the card. The IC chip preferably includes a central processing unit 310, memory 320 including a RAM 326, a EEPROM 324, a ROM 322, a timer 340, control logic 330, I/O ports 350 and security circuitry 360, which are connected together by a conventional data bus 390 or other conventional means.

Control logic 330 in the smart card provides sufficient sequencing and switching to handle read-write access to the card's memory through the input/output ports 350. CPU 310 in conjunction with control logic 330 can perform many different functions including performing calculations, accessing memory locations, modifying memory contents, and managing input/output ports. Some IC cards also include a coprocessor 370 for handling complex computations like cryptographic algorithms.

Input/output ports 350 are used for communication between the card and an interface device (IFD) which transfers information to and from the card. Timer 340 (which generates and/or provides a clock pulse) drives the control logic 330, CPU 310 and other components requiring a clock signal through the sequence of steps that accomplish functions including memory access, memory reading and/or writing, processing, and data communication. Security circuitry 360 (which is optional) preferably includes fusible links that connect the input/output lines to internal circuitry as required for testing during manufacture, but which are destroyed upon completion of testing to prevent later access.

The Static memory space is preferably mapped to memory locations in EEPROM 324 which are non-volatile. The Dynamic memory space is preferably mapped to RAM 326 which is volatile memory which has quick access. The Public memory space is also preferably mapped to RAM 326 which is volatile memory. The Dynamic data and Public data will be stored in different portions of RAM 326, while ROM is identified as a preferred non-volatile memory and EEPROM is identified as a preferred non-volatile, alterable memory. An operating system is preferably stored in ROM 322. One or more codelets are also preferably stored in ROM 322. The address table and one or more applications and codelets are preferably stored in EEPROM 324. Other types of memory could also be used with the same characteristics.

The foregoing merely illustrates the principles of the invention. It will thus be appreciated that those skilled in the art will be able to devise numerous systems

and methods which, although not explicitly shown or described herein, embody the principles of the invention and are thus within the spirit and scope of the invention.

The scope of the present disclosure includes any novel feature or combination of features disclosed therein either explicitly or implicitly or any generalisation thereof irrespective of whether or not it relates to the claimed invention or mitigates any or all of the problems addressed by the present invention. The applicant hereby gives notice that new claims may be formulated to such features during the prosecution of this application or of any such further application derived therefrom. In particular, with reference to the appended claims, features from dependent claims may be combined with those of the independent claims and features from respective independent claims may be combined in any appropriate manner and not merely in the specific combinations enumerated in the claims.

ANNEX A TO THE DESCRIPTION

BAKER & BOTTS LLP
30 ROCKEFELLER PLAZA
NEW YORK, NEW YORK 10112

TO ALL WHOM IT MAY CONCERN:

Be it known that WE, DAVID BARRINGTON EVERETT, STUART JAMES MILLER, ANTHONY DAVID PEACHAM, IAN STEPHENS SIMMONS, TIMOTHY PHILIP RICHARDS and JOHN CHARLES VINER, citizens of GREAT BRITAIN, whose post office addresses are 31 Ashdown Avenue, Saltdean, Brighton, East Sussex BN2 8AH; 9 Woodford Green, The Warren, Bracknell, Berks RG12 9YQ; 4 Lynwood, Groombridge, Tunbridge Wells, Kent TN3 9LX; The Elms, School Road, Broughton, Cambs PE17 3AT; 32 Craig Mount, Radlett, Herts WD7 7LW; and Hydes, Woodlands Lane, Windlesham; respectively, have invented an improvement in

MULTI-APPLICATION IC CARD WITH DELEGATION FEATURE

of which the following is a

SPECIFICATION

PRIORITY APPLICATIONS

This application claims priority to United States Provisional Application 60/046,514 filed on May 15, 1997, entitled "Design for a Multi Application Smart Card" and United States Provisional application 60/046,543 filed on May 15, 1997, entitled "Virtual Machine for a Multi Application Smart Card".

ANNEX A TO THE DESCRIPTION

BACKGROUND OF INVENTION

Integrated circuit ("IC") cards are becoming increasingly used for many different purposes in the world today. An IC card (also called a smart card) typically is the size of a conventional credit card which contains a computer chip including a microprocessor, read-only-memory (ROM), electrically erasable programmable read-only-memory (EEPROM), a random access memory (RAM), an Input/Output (I/O) mechanism and other circuitry to support the microprocessor in its operations. An IC card may contain a single application or may contain multiple independent applications in its memory. MULTOS™ is a multiple application operating system which runs on IC cards, among other platforms, and allows multiple applications to be executed on the card itself. The multiple application operating system present on the IC card allows a card user to run many programs stored in the card (for example, credit/debit, electronic money/purse and/or loyalty applications) irrespective of the type of terminal (i.e., ATM, telephone and/or POS) in which the card is inserted for use.

A conventional single application IC card, such as a telephone card or an electronic cash card, is loaded with a single application card and only executes that one application when inserted into a terminal. For example, a telephone card could only be used to charge a telephone call and could not be used as a credit/debit card. If a card user desires a variety of application functions to be performed by single application IC cards issued to him or her, such as both an electronic purse and a credit/debit function, the card

ANNEX A TO THE DESCRIPTION

user would be required to carry multiple physical cards on his or her person, which would be quite cumbersome and inconvenient. If an application developer or card user desired two different applications to interact or exchange data with each other, such as a purse application interacting with a frequent flyer loyalty application, the card user would be forced to swap multiple cards in and out of the card-receiving terminal during the transaction, making the transaction difficult, lengthy and inconvenient.

Therefore, it is beneficial to store multiple applications on the same IC card. For example, a card user may have both a purse application and a credit/debit application on the same card so that the user could select which type of payment (by electronic cash or credit card) to use to make a purchase. Multiple applications could be provided to an IC card if sufficient memory exists and an operating system capable of supporting multiple applications is present on the card.

The increased flexibility and power of storing multiple applications on a single card create new challenges to be overcome concerning the integrity and security of the information (including application code and associated data) exchanged between the individual card and the application provider as well as within the entire system when communicating information between applications.

For instance, the existence of multiple applications on the same card allows for the exchange of data between two applications, while one of the applications is being executed. As stated above, a frequent flyer loyalty program may need to be accessed during the execution of an electronic purse application. If data is passed

ANNEX A TO THE DESCRIPTION

between applications in an insecure manner, it may be possible for a third party monitoring the transaction to determine the contents of the transferred data or even other private data associated with one or both of the applications. Thus, it would be beneficial to provide an application architecture and memory organization which protects an application's data from being discovered by a third party when it is exchanged with other applications present on the IC card.

Accordingly, it is an object of the invention to provide an application architecture and memory organisation which provides for a secure data interaction between applications and allows multiple applications to be accessed while performing a desired task or function.

SUMMARY OF THE INVENTION

The present invention provides for a multiple application architecture for an IC card called an application abstract machine (AAM) and a method for implementing that architecture. The processing of multiple applications is accomplished by generating for at least one application (the "first application") a data memory space including at least two segments, a volatile memory segment and a non-volatile memory segment, commencing the execution of the first application's instructions; delegating or switching execution from the first application to the delegated application and in so doing, saving any data generated by the first application in the logical data memory space associated with the first application; executing the second application's instructions; retrieving the

ANNEX A TO THE DESCRIPTION

saved data and completing with this data the execution of the first application's instructions.

Additional delegation commands can be issued by the second application or other subsequent applications. The command delegated is interpreted by a delegated application in the same manner as a selection command being issued directly by a terminal and therefore each application performs the security functions at the same level as if a terminal is issuing the command.

The volatile memory segment can further be separated into public ("Public") and dynamic ("Dynamic") portions. Data can be exchanged between a plurality of applications and/or a terminal when stored in the Public region of the data memory. The Dynamic memory region can be used solely as temporary work space for the specific application being executed.

BRIEF DESCRIPTION OF THE DRAWINGS

Further objects, features and advantages of the invention will become apparent from the following detailed description taken in conjunction with the accompanying figures showing illustrative embodiments of the invention, in which

Fig. 1 is block diagram illustrating the data memory space segment and associated registers for an IC card application using the AAM organization;

ANNEX A TO THE DESCRIPTION

Fig. 2 is a block diagram illustrating the code memory and the data memory spaces for an IC card application using the AAM architecture;

Fig. 3 is a flow diagram illustrating the steps of performing a request for a delegation function by one application to another;

Fig. 4 is a flow diagram illustrating the steps of performing a return delegation control function for a delegate application to a delegator application;

Fig. 5 is a flow diagram illustrating the steps of performing an inquire delegator ID request of a delegation function;

Fig. 6 is a block diagram of an IC card chip which can be used as a platform in accordance with the invention; and

Figures 7A, 7B and 7C illustrate multiple delegation calls made between three applications.

Throughout the figures, the same reference numerals and characters, unless otherwise stated, are used to denote like features, elements, components or portions of the illustrated embodiments. Moreover, while the subject invention will now be described in detail with reference to the figures, it is done so in connection with the illustrative embodiments. It is intended that changes and modifications can be made to the described embodiments without departing from the true scope and spirit of the subject invention as defined by the appended claims.

ANNEX A TO THE DESCRIPTION

DETAILED DESCRIPTION OF THE INVENTION

The present invention provides for a method and apparatus for processing multiple application programs with associated data stored on an IC card which can be accessed and executed. An application stored on the card can be selected by a terminal, or other interface device, or another application. Each application program which is stored on the IC card when executed is allocated a memory space organized by the program's software code (instructions which are executed by a processor located on the IC card) and the associated data which the application stores and uses during execution of the program.

For example, a multi-application card may store a purse application, or an electronic money application, and a specific loyalty application such as a frequent flyer awards application. Each application has software code and associated data to support the execution of that software code. Each application is allocated a memory space when executed. In this example, there is interaction between the two applications stored on the card. For each dollar electronically spent to make a purchase, the user may be entitled to one frequent flyer mile which is stored and processed by the frequent flyer program. The purse application need not be aware of the specific loyalty program stored on the card, but instead may contain an instruction to communicate with any loyalty program stored on the card. The loyalty program will require input data representative of the amount of a particular electronic value so that it can update its own stored data of current frequent flyer miles for the user of the card.

ANNEX A TO THE DESCRIPTION

When two applications need to communicate during the same transaction, a system architecture is required to process both applications in an efficient and secure manner. One approach could be a windows type model where both applications could be running at the same time. Presently, however, IC card platforms are not powerful enough to simultaneously operate multiple programs efficiently. Also, transferred data may be exposed to unwanted third party access. The solution to this problem, provided by the current invention, which is described in greater detail below, is to selective interrupt the execution of applications in a secure manner. This allows the integrity of the applications' data to be maintained and allows the best utilization of the available memory space in the IC card.

An efficient architecture for processing multi applications in an IC card is termed an Application Abstract Machine (AAM) architecture and is described herein. The AAM Architecture applies to any platform independent of its hardware and enables developers to write applications to store on the IC cards which are portable across many different types of platforms (e.g., IC cards built by different manufacturers with different processor configurations) without the need for knowledge about the specific hardware of the platform.

An application abstract machine (AAM), a term for the memory allocation and organization for the data stored and used by each application, is created for each application stored on the IC card which is executed by the processor on the card. In order to ensure data integrity and security when data is transferred between applications which

ANNEX A TO THE DESCRIPTION

are executed on the IC card, only one application on the IC card is allowed to be executed at a time. Each application has a data memory space which is virtually allocated and mapped onto the physical memory addresses available in the IC card memories. Data is then passed between two or more applications within a specified memory location and in a manner consistent with transferring data to an external terminal or device with which the IC card is securely interacting. At a general level, each AAM space created for each application being executed includes two separate address spaces, one for the program code itself and one for the program data which is stored and/or used by the application. The program data address space is effectively divided into three segments: a Static segment, a Dynamic segment and a Public segment which are described in more detail in conjunction with Figure 1. As stated above, the Static, Dynamic and Public segments are logically mapped to the physical memory; they are virtual memory segments as opposed to physical memory segments. The AAM data address space is preferably addressed and processed using seven different address registers and two control registers.

Figure 1 shows an illustrative diagram of a logical data space allocation 101 created for an application used in conjunction with the present invention. The AAM data portion 101 includes a Static data space 103, a Public data space 105 and a Dynamic data space 107. Also shown are a series of address registers: the Static base address register 109, the Static top address register 111, the Public base address register 113, the Public top address register 115, the Dynamic base address register 117, the Dynamic top address register 121 and local base address register 119 which serves as a local stack

ANNEX A TO THE DESCRIPTION

frame pointer in the Dynamic data space when the application is being executed. The address registers can contain physical memory addresses but preferably contain offset addresses for the various data address spaces in order to be hardware independent. An example of the overall address space is 64K bytes, although the size varies with the applicable platform and the available memory size. The registers can also be considered pointers or can be any other conventional addressing mechanism.

Within the allocated AAM data space 101, the Static portion of the memory is non-volatile which is not erased after power is removed from the IC card (such as EEPROM), the Dynamic space is volatile (such as RAM) which may be erased after power is removed from the card and the Public space is also volatile (such as RAM). An IC card can receive power from a terminal after it is interfaced into the terminal. Although an IC card may contain a battery to maintain some power for memory and circuitry, volatile memory will typically be erased after the IC card is removed from its power source.

The defined AAM data space has bytes in each segment which are contiguous, so that applications can perform pointer and offset arithmetic. For example, if the segment addresses "1515" and "1516," or any other pair of sequential numbers, are both valid and are present within the same segment, then they address adjacent bytes. This allows offset values stored in registers to determine the location of a desired memory address. The segment address of the first byte of the Static segment is zero, so that the segment address of a given location within the Static region is equal to its offset.

ANNEX A TO THE DESCRIPTION

Pointers to other specific regions of the Static data area can be stored in the Static data because the Static region is non-volatile. For example, if the card user's name is stored in the Static memory of a credit/debit application, the application will know the card user's name will always be stored in the 5th memory location above the starting point for the Static portion of memory. The location can be noted as SB[5] or the 5th byte above the Static Bottom. Since the Static memory is non-volatile, it will not be erased after each transaction and the application will always know of its location relative to the Static segments' address registers.

On the other hand, the segment address of any location in the Dynamic or Public segments is not always equal to a particular offset from the beginning of the respective segment because the contents of those segments change for each operation. The fourth location in the Dynamic segment will be different for each operation performed by the application. The address of a memory location of Dynamic or Public segment is fixed preferably only for the duration of one command-response pair operation. Because segment addresses in Dynamic or Public are not fixed, MULTOS Executable Language (MEL)™ instructions (or any other program instructions) cannot refer to data using only segment addresses. Instead, a tagged address preferably is used to identify data which is to be retrieved, manipulated, transferred and/or stored with the IC card system.

A tagged address is a nineteen bit value consisting of a three bit tag (address register number) and a sixteen bit offset. Each of the seven address registers for

ANNEX A TO THE DESCRIPTION

the AAM data space contain a segment physical address. For instance, the address registers SB 109 and ST 111 point to the boundaries of the Static, the address registers PB 113 and PT 115 point to the boundaries of the Public and the address registers DB 117 and DT 121 point to the boundaries of the Dynamic. For each segment, the top register points to the byte immediately after the last valid byte. For example, the last valid byte of the Static is ST[-1]. Register LB functions as a stack frame pointer. It points to a location in the Dynamic segment to indicate a specific byte of local data for the currently executing application.

Referring to Figure 1, the allocated Static segment 103 contains the application's non-volatile data. Static data includes data which is associated with each application for every transaction such as the card user's name, account number, PIN value and address. Static data also includes variable data which is stored for use in future transactions using the application. For example, in a purse transaction, the electronic value data would be read from the Static segment and later saved in the Static segment at the end of the transaction. Additionally, transaction information data or available credit limits in the case of a credit/debit application would be stored in Static data.

The Static data is addressed using register SB (Static Base) and the register ST (Static Top) as offset registers. These registers contain the offset value from a physical address in a memory on the IC card. The individual memory location is then further offset from these starting points such as SB[3] or ST[-5]. SB is defined as zero and ST is equal to the size of the application's Static data which is set when the

ANNEX A TO THE DESCRIPTION

application is loaded onto the IC card. The multiple application operating system ensures that no other application can read or write the data stored in the Static segment of a particular application. Using current technology, the Static segment is preferably mapped onto an EEPROM (Electrically Erasable Programmable Read-Only Memory) which is non-volatile.

The Dynamic segment 107 contains the application's volatile or temporary data. Dynamic data includes data which is temporarily used during the execution of an application such as intermediate values used in calculations or working variables. For example, a purse application may temporarily store the value of a transaction in order to reduce the amount of the value in the purse. The temporary data is used much like conventional computer programs use RAM to perform their assigned operations. The Dynamic segment preferably is divided into two parts, the session data portion and the stack data portion. The size of the session data is a constant for each application and is determined when the application is loaded. The stack holds variable data which is unique to the particular transaction being executed. The stack data portion stores data in a last-in-first-out manner. The stack is initially empty, but expands and contracts during execution of the application.

The Dynamic data is addressed from the register DB 117 to register DT 121. Register LB 119 serves as a local stack frame pointer to particular memory locations in the Dynamic segment for delegate commands or function calls. Register LB 119 is used to address the topmost frame, that of the currently executing function's

ANNEX A TO THE DESCRIPTION

session data. Register DT 121 serves as an address offset for the stack pointer. A one byte data item at the top of the stack is addressed as DT[-1], the next byte below is addressed by DT[-2], and so on. A push operation increments the relative value of DT for each item on the stack and a pop operation decrements the relative value of DT for each item on the stack. For example, a data element located at DT[-5] will be located at DT[-6] after an additional data item is placed on the stack.

When an application is being executed, the Dynamic segment created for that application also contains the application's session data which is used in performing the assigned task(s) or operation(s). The multiple application operating system ensures that no other application can read or write the data stored in the Dynamic segment of a particular application. The session data is set to zero upon the start of the execution of the application. Stack data will be saved in the stack if the application delegates a task or operation to another application.

A delegation function occurs when one application selects another application to process a command instead of processing the command itself. An example of a delegation function occurs when a delegator application receives a command that it does not recognize or is not programmed to process. The selected application should not reject the command and provide an error response to the interface device (IFD), but instead should pass the command to the appropriate receiver, or delegated application. In order to perform a delegation, the delegator calls the Delegate primitive. The Delegate primitive is a subroutine recognized by the multiple application operating system which

ANNEX A TO THE DESCRIPTION

is executed when the operating system interprets the Delegate instruction. Primitives can be stored as part of the operating system itself, loaded as a separate routine when the operating system is installed. Primitives are preferably written in machine executable language so that they can be executed quickly although they could be written in a higher level language. When a Delegate command is executed, execution of the delegating application is suspended, and the delegated application is executed instead. The delegated application then generates its own data memory space according to the AAM architecture. The data stored in the Public memory space of the first application (stored in RAM) is sent to the Public memory space of the second application (which could be physically the same memory but is allocated separately for each application) so that data can be passed between the applications. The Dynamic memory space is also shared although data is saved in a stack for the delegator and the other portions initialized before the delegated application is executed because the Dynamic data is secret.

In most cases, the delegated application processes the command exactly as though the command has arrived directly from an interface device. When the delegated application has finished processing the command, and has written a response into the allocated Public memory segment, it exits as normal. The delegator then resumes execution at the instruction address following the executed instruction which called the Delegate primitive. The response generated by the delegated application is retrieved or accessed from the allocated Public memory space. The delegator application may simply

exit in turn, thus sending the response to the IFD, or may carry out further processing before exiting.

Another example of a delegation operation occurs when two applications need to share data. If an application A always returns a data item N when processing a command B, then another application which also returns data item N in response to a command can delegate the function B to application A in order to reduce the need for duplicate codes stored on the IC card. For example, if a PIN needs to be checked before an application is executed, an application stored on the card can delegate the "retrieve PIN function" to a PIN application which returns a stored universal PIN for the card.

Preferably, a new session begins whenever the IFD, e.g. a terminal, successfully selects an application, even if the application has been previously selected during the transaction. For example, if a card user goes to a terminal and transfers twenty dollars of electronic cash using a purse application, charges thirty dollars using a credit/debit application and then transfers ten dollars using the purse application again, three separate sessions will have occurred even though only two applications were used during the entire transaction. Each time an application delegates a task or function to another application, the delegated application treats the delegate function as if the IFD devices had selected the application to perform the task or function. However, performing a delegation function as described below has a different effect on session data.

The following examples will help explain when the session data is initialized (i.e., erased) versus when it is saved to be used in further operations. If application A is selected by an IFD device, and receives commands X, Y and Z from the terminal, application A may delegate all three commands to application B. For example, delegations may occur in response to delegation commands in the program code. Both applications A and B will have their session and stack data in their respective Dynamic segments initialized (set to zero) when they receive command X, but the stack will not be initialized when they receive the subsequent commands Y and Z.

In a second example, application A is selected, and receives commands X, Y and Z from the terminal. Application A processes X itself, but delegates Y and Z to application B. Application A will have its session and stack data initialized when it receives X, but not when it receives the subsequent commands Y and Z. Application B will have its session and stack data initialized when it receives Y, but not Z.

One example of a use of session data is to support the use of a session Personal Identification Number (PIN). The application could reserve one byte of session data to support the PIN-receiving flag. On receiving the PIN check command, the selected delegated application could update the flag as follows: if the PIN command is received and the inputted PIN is equal to the stored pin, then it will set the session data DB[0] to 1. If not, the application will check if the PIN flag is already set by checking the value in DB[0]. In either of the above cases, the application will process the rest of the commands in the session because the PIN has been verified. If neither of the cases is

ADD A TO THE DESCRIPTION

true, then the application will not process the command because the PIN is not proper. The PIN checking function could be a delegated function from the selected application to a PIN checking application.

The Public segment 105 is used for command and response data being passed between an IFD and an application. During a delegate command, the Public segment contains the data passed between two applications, the delegator (the application initiating the delegation) and the delegated application (the application which performs the delegated function). An application may also use the Public segment as a further temporary working storage space if required. The Public data is addressed using offsets stored in register PB 113 as a starting address, to register PT 115 as an ending address. Register PB 113 and Register PT 115 are fixed for the duration of a command-response pair being initiated by the IFD or delegator. Public data can include data inputted into or supplied by a terminal such as a transaction amount, vendor identification data, terminal information, transmission format or other data required or used by an application resident on the IC card. Public data can also include data which is to be transmitted to an IFD device or other application such as an electronic dollar value, card user information transmission format or other data required or used by the terminal or other delegated application.

The multiple application operating system ensures that the data stored in the Public segment remains private to the application until the application exits or delegates. Preferably, the data in the Public segment is then made available to other

ANNEX A TO THE DESCRIPTION

entities as follows: (1) if the application delegates, the whole of the Public segment becomes available to the delegated application; (2) if the application exits, and is itself delegated by another, the whole of the Public segment becomes available to the delegator; or (3) if the application exits, and is not itself delegated, then a portion of the Public segment containing the I/O response parameters and data are made available to the IFD.

An application may write secret data into the Public memory segment during execution of the application, but the application must make sure it overwrites the secret portion of the Public segment before delegating or exiting. If the application abnormally ends (abends), then the operating system on the IC card preferably overwrites all of the data in the Public segment automatically so that no unwanted entities can have access to the secret data. If the MULTOS carrier device (MCD) is reset, the operating system overwrites data in the Public segment automatically, so that no secret data is revealed. A portion of the Public memory segment is also used as a communications buffer. The I/O protocol data and parameters are preferably stored at the top of the Public memory space. In another preferred embodiment, the top seventeen bytes are reserved for the communications protocol between the IFD device and the IC card application. However, additional or less bytes can also be used depending upon the particular application and operating system being utilized.

The spaces shown between the memory segments in Figure 1 will vary depending upon the specific application and commands being processed. There could be

ANNEX A TO THE DESCRIPTION

no memory space between the memory segments so that the memory segments are contiguous.

Figure 2 shows an extended illustration of the AAM implemented architecture. Data memory space 201 includes the three segments Static, Public and Dynamic as previously described. Code memory space 203 contains the program instructions for an application stored on the IC card. The application instructions are preferably stored in an executable form which can be interpreted by the resident operating system but can also be stored in machine executable form. Instruction 205 is stored at one location in the code memory space 203. Additional instructions are stored in other locations of memory space 203. Two additional registers 207 and 209 are used in the AAM architecture. A code pointer (CP) register 207 indicates the particular code instruction to be next executed. In the figure, the register indicates, e.g., through an offset or pointer means, that instruction 205 is the next to be executed. Condition Control Register 209 contains eight bits, four of which are for use by the individual application and four of which are set or cleared depending upon the results of the execution of an instruction. These condition codes can be used by conditional instructions such as Branch, Call or Jump. The condition codes can include a carry bit, an overflow bit, a negative bit and a zero bit.

All address and control registers are set to defined values prior to executing the selected or delegated application. The values are set either when the application is first loaded onto the card and the size of the code and non-volatile data can

 ANNEX A TO THE DESCRIPTION

be ascertained or at the moment when the application passes control to the application. When the application is loaded, SB is set to zero and ST is equal to the number of bytes in the application's Static database. The other address registers are initialized when the application is given control. CP 207 is set to zero and all eight bits in CCR 209 are cleared at the start of executing the application.

A communications interface mechanism is present between the IFD and an application which includes the use of the Public data segment as a communications buffer for command-response parameters. A command-response parameter means an application is given a command to perform and returns a response to the entity issuing the command. Applications interact with an IFD by receiving commands, processing them and returning responses across the IFD-Application Interface. When an application has completed executing a command, the application will place the response into the Public segment starting at PB[0] which can be read by the IFD device and will set the proper interface parameters in the reserved Public space relative to PT[0].

While an application can be called directly from an IFD and return a response directly to an IFD, it can also delegate a request to another application where appropriate. The subsequently-called application will then process the request on behalf of the first application. The delegation can be directly in response to a received command in which the delegator acts as a controller for delegating commands or subcommands to other appropriate applications. Alternatively, the delegated command can be embedded in an application's code which delegates control of the processor when the first

ANNEX A TO THE DESCRIPTION

application needs to interact with another application during its execution, such as updating frequent flyer miles or verifying a PIN.

Figure 3 shows a flow chart of the steps which are performed when a delegate request is executed. Step 301 sets the parameter named delegator_application_id (delegator ID) to be equal to the selected_file.application_id (selected ID). The selected ID indicates the current application which is selected and which is currently being executed. The delegator ID indicates the application which delegates a function to another delegated application stored on the IC card. Step 303 then pushes (stores) the delegator ID onto the top of the delegate_id_stack (delegate stack). The data referenced in the Dynamic portion of allocated memory is saved so that the current application can complete its execution after the delegated function is complete. Data which is to be shared with the delegated application is referenced in the Public portion of allocated memory. The delegate stack is preferably stored outside of an application's AAM memory space and keeps track of which applications have delegated functions. Each application is suspended when it delegates a function so the delegate stack can act in a Last-In-First-Out (LIFO) manner so that if a number of applications are suspended due to delegation requests, the proper application is started in the right order. The delegate stack thus keeps track of which application was the last delegator when multiple layered delegation functions are performed. The delegate stack preferably operates in a LIFO manner although different stack schemes could be used as appropriate.

ANNEX A TO THE DESCRIPTION

Step 305 then sets the selected ID to the delegate_request.delegate_ application_id (delegate ID) value. This step selects the application which will be called to perform the delegated function or functions. The identities of the delegated application can be specifically called by the delegator application or a particular function can be matched up with an application in a look up table. For example, a PIN match operation may be delegated to different applications depending upon which applications are present on the card. Step 307 then sets the application_command parameter to the value stored in the delegate_request.application_command parameter. This step specifies the command to be delegated to the delegate application. Applications typically have the ability to process many different commands. Alternatively, the entire application could be executed to perform one or more functions. The delegator application can choose which command it is delegating to another application. Step 309 then sends the application_command to the AAM operating system for execution by the delegatee application. The delegator application is then suspended (or interrupted). Any data that is required to pass between the applications is transferred via the Public memory space.

Figure 4 is a flow chart of the steps for performing a "return delegation control" command by the delegatee application. This command is executed by the operating system when a delegated application has completed its delegated function. Step 401 gets application_responses from the Public memory space of the delegated AAM. The response data is passed in the Public memory segment of the delegatee AAM. Step 403 then sets the delegate_response.status variable to a success condition. This

ANNEX A TO THE DESCRIPTION

means that a delegation operation has been successfully completed. Step 405 sets the delegate_ response.application_responses parameter to the application_responses values which were stored in the Public segment of the delegatee application.

Step 407 sets the delegate_response.delegate_application_id parameter to selected_file.application_id (the delegatee application ID). Step 409 pops the top (i.e., reads the last data stored in the stack) delegate_application_id from the delegate_id_stack. This information indicates the identity of the delegator application for the command which was just delegated and completed by the delegated application. Step 411 sets the select_file.application_id value to the delegator_application_id value. This selects the delegator application which was identified from the delegate ID stack as the current application which will resume running. The Dynamic data for the delegator application will be retrieved for the delegator application from its stored location so that the application will continue to execute where it left off with all data intact but will also have the response information from the delegated function. In step 413, the delegate_response data is sent to the current application for further processing. The response data is passed through the Public data space which could be the same physical RAM memory location because all applications share the physical volatile memory space.

Figure 5 shows a flow chart of the steps involved for inquiring about a delegator ID when a delegate command is received by a delegated application. The delegated application may need to know the identity of the delegator because it may perform operations differently for different delegator applications. For example, an

ANNEX A TO THE DESCRIPTION

airline loyalty program may need to know if awarded frequent flyers will be based on actual dollars processed or a lump sum award for some other activity such as performing a bill payment operation. This information could be passed to the delegated application as a variable or could be ascertained using an inquiry. The delegator inquiry operation could be implemented as a primitive as previously described.

Step 501 receives the delegator_id_enq_request from the AAM operating system. The request is used to identify the identity of the delegator. Step 503 checks if the delegate_id_stack is empty. If the stack is empty, then no delegation operations have occurred and no applications have been suspended. Thus step 511 sets the delegator_id_enq_response.status parameter to a failure indicator. Step 513 then sets the value of delegator_is_enq_request.error_cause to a value indicating "no delegator application." There is no delegator application. The process then continues with step 509.

If the delegate_id_stack is not empty, than one or more delegations have occurred. In that case, step 505 sets the delegator_id_enq_response.status parameter to a value indicating "success". Step 507 then sets the delegator_id_enq_response.delegator_application_id parameter to the value stored in delegate_id_stack.delegator_application_id. This sets the inquiry response to indicate the delegator application ID at the top of the stack. As explained above, the stored data at the top of the stack indicates the last delegator application to call a delegate function. Step 509 then sends the

ANNEX A TO INTERNATIONAL

delegator_id_enq_ response back to the AAM operator system which delivers the information to the application or IFD entity requesting the information.

Figure 6 shows an example of a block diagram of an integrated circuit located on an IC card chip which can be used in conjunction with the invention. The integrated circuit chip is located on a chip on the card. The IC chip preferably includes a central processing unit 601, a RAM 603, a EEPROM 605, a ROM 607, a timer 609, control logic 611, I/O ports 613 and security circuitry 615, which are connected together by a conventional data bus 617 or other conventional means.

Control logic 611 in the smart card provides sufficient sequencing and switching to handle read-write access to the card's memory through the input/output ports 612. CPU 601 in conjunction with control logic 611 can perform many different functions including performing calculations, accessing memory locations, modifying memory contents, and managing input/output ports. Some IC cards also include a coprocessor for handling complex computations like cryptographic algorithms. Input/output ports 613 are used for communication between the card and an IFD which transfers information to and from the card. Timer 609 (which generates and/or provides a clock pulse) drives the control logic 611, CPU 601 and other components requiring a clock signal through the sequence of steps that accomplish functions including memory access, memory reading and/or writing, processing, and data communication. Security circuitry 615 (which is optional) preferably includes fusible links that connect the input/output lines to internal circuitry as required for testing during manufacture, but

ANNEX A TO THE DESCRIPTION

which are destroyed upon completion of testing to prevent later access. The Static memory space is preferably mapped to memory locations in EEPROM 605 which is non-volatile. The Dynamic memory space is preferably mapped to RAM 603 which is volatile memory which has quick access. The Public memory space is also preferably mapped to RAM 603 which is volatile memory. The Dynamic data and Public data will be stored in different portions of RAM 603, while RAM is identified as a preferred non-volatile memory and EEPROM is identified as a preferred volatile memory. Other types of memory could also be used with the same characteristics.

Figures 7A, 7B and 7C illustrate an example of a delegation function being performed in order to process multiple applications on an IC card. Figure 7A shows a first application being executed as denoted with a double ringed circle 701. At some point during the execution of the first application, a delegation function 702 is called to delegate an operation to the second application which is indicated by circle 703. Also shown in Figure 7A is an empty delegator ID stack 705. Since the stack is empty, there is no data associated with it and it is shown only for illustrative purposes.

The multiple application operating system receives the delegate command and interrupts the execution of the first application 701 and gives control of the integrated circuit to application 703 as shown in Figure 7B. The execution of the second application 703 is illustrated with a double ringed circle. The term "gives control" means that the microprocessor and other circuitry on the card will process the instructions and allocate memory space for the application which is delegated. When the delegate command is

ANNEX A TO THE DESCRIPTION

processed, the delegator ID 707 is placed on top of the stack 705. The delegator ID stack is operated in a LIFO manner. Also shown in Figure 7B is a third application 709 resident on the card. At some point during the execution of the second application, a delegate function 711 is called to delegate the operation to the third application.

The multiple application operating system receives the delegate command 711 shown in Figure 7B interrupts the execution of the second application 703 and gives control of the integrated circuit to the third application 709 as shown in Figure 7C. When the delegate command is processed, the delegator ID 713 of the second application is pushed onto the delegator ID stack 705. The delegator ID 707 of the first application whose execution is still interrupted is pushed down in the stack consistent with a LIFO stack management. Thus when the third application has finished its execution, the delegator ID at the top of the stack is popped to indicate that execution of the second application should be resumed first. The delegator ID 707 from the first application will then be at the top of the stack so that when the second application is finished executing, the first application will resume its execution.

Additional applications can be managed by the delegator ID stack in a similar manner. By interrupting the execution of the applications when a delegate command is processed and keeping track of the order of delegations, the security and integrity of the data for each individual application can be maintained which is important because IC cards will store data for applications which is private to the card user such as account numbers, social security number, address and other personal information.

 ANNEX A TO THE DESCRIPTION

The foregoing merely illustrates the principles of the invention. It will thus be appreciated that those skilled in the art will be able to devise numerous apparatus, systems and methods which, although not explicitly shown or described herein, embody the principles of the invention and are thus within the spirit and scope of the invention.

ANNEX A TO THE DESCRIPTION

WE CLAIM:

1. An integrated circuit card comprising:

a microprocessor; a volatile memory coupled to said microprocessor; a non-volatile memory coupled to said microprocessor; and a plurality of applications stored in said non-volatile memory, wherein upon execution of each said application, said microprocessor allocates for each said executing application an associated data memory space comprising at least a volatile memory segment for referencing temporary data and a non-volatile memory segment for referencing static data; and further comprising means for delegating the performance of a function from a first executing application to a second executing application.

2. The integrated circuit card of claim 1, wherein said non-volatile memory segment is divided into at least two regions, including a public region and a dynamic region.

3. The integrated circuit card of claim 2, wherein said public region is used to share data between said first and second applications.

4. The integrated circuit card of claim 2, wherein said dynamic region is used to reference temporary data utilized during an application's execution.

ANNEX A TO THE DESCRIPTION

5. The integrated circuit card of claim 1, further comprising at least one register coupled to said microprocessor which is used to determine the starting locations of each of said segments.

6. The integrated circuit card of claim 5, further comprising at least one register coupled to said microprocessor which is used to determine the top locations of each of said segments.

7. The integrated circuit card of claim 6, further comprising at least one register coupled to said microprocessor which is used as a local dynamic pointer.

8. The integrated circuit card of claim 1, wherein each said application comprise a plurality of program instructions and wherein at least one of said program instructions when executed causes said memory referenced by said volatile memory segment to be accessed.

9. The integrated circuit card of claim 1, wherein said volatile memory segment references RAM and said non-volatile memory segment references EEPROM.

10. A method for processing a plurality of applications stored in a memory of an integrated circuit:

ANNEX A TO THE DESCRIPTION

selecting a first application for execution;

allocating a data space for said first application including at least two memory segments comprising a volatile memory segment for referencing temporary data and a non-volatile memory segment for referencing static data;

executing said first application, interrupting execution of said first application and saving data referenced by said volatile memory segment;

executing a second application;

utilizing said saved data from said volatile memory segment for execution of said first application; and

completing said execution of said first application.

11. The method of claim 10, wherein said first application's identity is stored in a data stack during said delegation step.

12. The method of claim 11, wherein said data stack is accessed following said completion of said second application.

13. The method of claim 12, further including the step of inquiring said first application's identity by accessing said delegator stack.

14. The method of claim 10, wherein said non-volatile memory segment is divided into at least two regions, including a public region and a dynamic region.

ANNEX A TO THE DESCRIPTION

15. The method of claim 14, wherein said public region is used to share data between said first application and said second application.

16. The method of claim 14, wherein data referenced by said dynamic region is utilized during the execution of said first application.

17. The method of claim 10, further including the step of allocating a second data space including at least two memory segments for said second application.

18. The method of claim 17, wherein said second data space's segments comprise a volatile memory segment for referencing temporary data and a non-volatile memory segment for referencing static data.

19. The method of claim 18, wherein said second application's non-volatile segment is divided into at least two regions, including a public region and a dynamic region.

20. The method of claim 19, wherein said second application's public region is used to share data between said first and second applications.

21. The method of claim 19, wherein said data referenced by second application's dynamic region is utilized during said execution of said second application.

ANNEX A TO THE DESCRIPTION

22. The method of claim 10, further including the step of delegating use of said microprocessor from said second application to a third application stored on said IC card.

23. The method of claim 22, wherein a third data space for said third application is allocated which includes a volatile memory segment for referencing temporary data and non-volatile memory segment for referencing static data, wherein said third application's volatile segment includes a public and dynamic portion.

24. An apparatus for processing a plurality of applications stored in a memory of a single integrated circuit card comprising:

means for allocating a data space comprising at least a non-volatile memory segment for referencing static data and a volatile memory segment for referencing temporary data; means for executing a first application; means for interrupting execution of said first application, means for saving data from at least a portion of said volatile memory segment; and means for executing a second application; means for retrieving said saved data; and means for completing said execution of said first application.

25. The apparatus of claim 24, further including means for storing said first application's identity on a data stack.

APPENDIX A TO THE DESCRIPTION

26. The apparatus of claim 25, further including means for inquiring of said first application's identity.

27. The apparatus of claim 24, wherein said first application's non-volatile memory segment is divided into at least two regions, including a public region and a dynamic region.

28. The apparatus of claim 27, wherein said public region references random access memory.

29. The apparatus of claim 27, wherein said dynamic region references random access memory.

30. The apparatus of claim 24, further including means for allocating a second data space including at least two segments for said second application.

31. The apparatus of claim 30, wherein said second data space includes a volatile memory segment for referencing temporary data and a non-volatile memory segment for referencing static data.

ANNEX A TO THE DESCRIPTION

32. The apparatus of claim 31, wherein said second data space's non-volatile segment is divided into at least two regions, including a public region and a dynamic region.

33. The apparatus of claim 32, wherein said public region references random access memory.

34. The apparatus of claim 32, wherein said dynamic region references random access memory.

35. The apparatus of claim 24, further including means for delegating operation of said IC card from said second application to a third application stored on said IC card.

36. The apparatus of claim 35, wherein a third data space for said third application is allocated which includes a volatile memory segment for referencing temporary data and non-volatile memory segment for referencing temporary data, wherein said third application's volatile memory segment includes a public and dynamic portion.

37. A system for processing a plurality of applications stored on an IC card comprising:

a non-volatile memory coupled to a databus;

ANNEX A TO THE DESCRIPTION

a volatile memory coupled to said databus;

a first and second application program stored in said non-volatile memory, wherein each application has an associated identifier;

a data stack accessible by said databus for storing said applications' identifier if said application is interrupted during its execution;

processor means for executing instructions from said application programs wherein said processor means allocates a data memory space for said application which is being executed and said data memory space is mapped to at least one address in said non-volatile memory and at least one address in said volatile memory; and

wherein said processor means interrupts said first application at least once during its execution to execute said second application.

38. The system of claim 37, wherein data memory space comprises at least a volatile memory segment for referencing temporary data stored in said volatile memory and a non-volatile memory segment for referencing static data stored in said non-volatile memory.

39. The system of claim 37, further including means for storing said first application's identity on a data stack.

40. The system of claim 39, further including means for inquiring of said first application's identity.

ANNEX A TO THE DESCRIPTION

41. The system of claim 38, wherein said first application's non-volatile memory segment is divided into at least two regions, including a public region and a dynamic region.

42. The system of claim 41, wherein said public region references random access memory.

43. The system of claim 41, wherein said dynamic region references random access memory.

44. The system of claim 37, further including means for allocating a second data space including at least two segments for said second application.

45. The system of claim 44, wherein said second data space comprises at least a volatile memory segment for referencing temporary data and a non-volatile memory segment for referencing static data.

46. The system of claim 45, wherein said second data space's non-volatile segment is divided into at least two regions, including a public region and a dynamic region.

ANNEX A TO THE DESCRIPTION

47. The system of claim 46, wherein said public region references random access memory.

48. The system of claim 46, wherein said dynamic region references random access memory.

49. The system of claim 37, further including means for delegating use of said processor means from said second application to a third application stored on said IC card.

50. The system of claim 49, wherein a third data space for said third application is allocated which includes a volatile memory segment for referencing temporary data and non-volatile memory segment for referencing temporary data, wherein said third application's volatile memory segment includes a public and dynamic portion.

51. An integrated circuit card comprising:

a plurality of applications and a microprocessor for controlling execution of said applications wherein execution of at least one first application is interrupted and execution is transferred to another second application, further comprising means for sharing data by said first and second applications and means for resuming execution of said first application at the appropriate location at least after completion of execution of said second application.

 ANNEX A TO THE DESCRIPTION

52. The integrated circuit card of claim 51, further comprising means for allocating a data memory space comprises at least a volatile memory segment for referencing temporary data stored in said volatile memory and a non-volatile memory segment for referencing static data stored in said non-volatile memory.

53. The integrated circuit card of claim 51, further including means for storing said first application's identity on a data stack.

54. The integrated circuit card of claim 53, further including means for inquiring of said first application's identity.

55. The integrated circuit card of claim 52, wherein said first application's non-volatile memory segment is divided into at least two regions, including a public region and a dynamic region.

56. The integrated circuit card of claim 55, wherein said public region references random access memory.

57. The integrated circuit card of claim 55, wherein said dynamic region references random access memory.

ANNEX A TO THE DECLARATION

58. The integrated circuit card of claim 52, further including means for allocating a second data space including at least two segments for said second application.

59. The integrated circuit card of claim 58, wherein said second data space comprises at least a volatile memory segment for referencing temporary data and a non-volatile memory segment for referencing static data.

60. The integrated circuit of claim 58, wherein said second data space's non-volatile segment is divided into at least two regions, including a public region and a dynamic region.

61. The integrated circuit of claim 58, wherein said public region references random access memory.

62. The integrated circuit card of claim 60, wherein said dynamic region references random access memory.

63. The integrated circuit card of claim 51, further including means for delegating use of said processor means from said second application to a third application stored on said IC card.

ANNEX A TO THE DESCRIPTION

ABSTRACT OF THE DISCLOSURE

A multi-application IC card which processes two or more applications using an Application Abstract Machine architecture. The AAM architecture only allows one application to be executed at a time and allows for shared processing by performing a delegation function to a second application. A data space for each application is allocated when the application is selected to be executed. The data space includes a volatile and non-volatile region. The delegation function temporarily interrupts the execution of the first application, saves the temporary data of the first application, shares any data needed with the second application and the second application is executed until the delegated task is competed. The first application then retrieves the saved data and completes its execution. A delegator stack is used to keep track of the delegator's identity when multiple delegations occur. The AAM model allows for a high level of security while transferring data between applications.

CLAIMS

1. An integrated circuit card comprising:

a read-only memory;

an alterable memory;

a microprocessor coupled to said read-only and alterable memory;

an operating system stored in said read-only memory, wherein said operating system is executed by said microprocessor;

at least one application stored in said alterable memory; and

a codelet stored in said read-only memory and comprising at least one non-native program instruction.

2. The integrated circuit card of claim 1, wherein said alterable memory comprises an EEPROM.

3. The integrated circuit card of claim 1 or claim 2 wherein one of said application's program instruction calls said codelet.

4. The integrated circuit card of any preceding claim, further comprising an address table for storing said memory address and identifier of said stored codelet.

5. The integrated circuit card of claim 4, wherein said address table is stored in said alterable memory.

6. The integrated circuit card of claim 4 or claim 5, wherein said operating system executes said codelet by looking up said codelet's memory address in said address table using said codelet's identifier.

7. The integrated circuit card of any preceding claim, further including a primitive stored in said read-only memory.

8. The integrated circuit card of claim 7, wherein said primitive verifies said codelet's presence on said IC card.

9. The IC card of claim 8, further including a register indicative of said codelet's presence.

10. The integrated circuit card of any of claims 7 to 9, wherein said codelet executes said primitive.

11. The integrated circuit card of claim 10, wherein said execution of said primitive uses an address table look up to determine said primitive's memory location.

12. A method for efficiently storing programming instructions in a microprocessor system including read-only memory and alterable memory comprising the steps of:

storing an operating system in said read-only memory;

storing at least one application written in a non-native computer language in said alterable memory;

storing a codelet comprising program instructions written in a non-native computer language in said read-only memory; and

storing said codelet's address location in an address table;

wherein said codelet is accessed by said operating system using said address table.

13. The method of claim 12, wherein said alterable memory comprises EEPROM.

14. The method of claim 12 or 13, wherein one of said application's program instructions calls said codelet.

15. The method of any of claims 12 to 14, wherein said address table is stored in said alterable memory.

16. The method of any of claims 12 to 15, wherein said operating system executes said codelet by looking up said codelet's memory address in said address table.

17. The method of any of claims 12 to 16, wherein said address table is stored in said read-only memory.

18. The method of any of claims 12 to 17, further including the step of storing a primitive in said read-only memory.

19. The method of any of claims 12 to 18, wherein said non-native computer language of said at least one application is said non-native computer language of said codelet.

20. The method of any of claims 12 to 19, further including the step of verifying the presence of said codelet on said IC card.

21. A system for efficiently storing program instructions in a microprocessor based system comprising:

A read-only memory storing an operating system and at least one codelet, wherein said codelet comprises program instructions written in a non-native programming language requiring interpretation by said operating system; and

An alterable memory storing at least one application comprising program instructions and data and an address table;

wherein said memory address and identifier of said codelet is stored in said address table and said codelet is accessed by said at least one application during said application's execution.

22. The system of claim 21, wherein said codelet's program instruction utilizes said application's data.

23. The system of claim 21 or 22, further including a means for verifying said codelet's presence in said read-only memory.

24. The system of any of claims 21 to 23, wherein said system resides on an integrated circuit card.

25. The system of any of claims 21 to 24, wherein said operating system executes said codelet by looking up said codelet's memory address in said address table using said codelet's identifier.

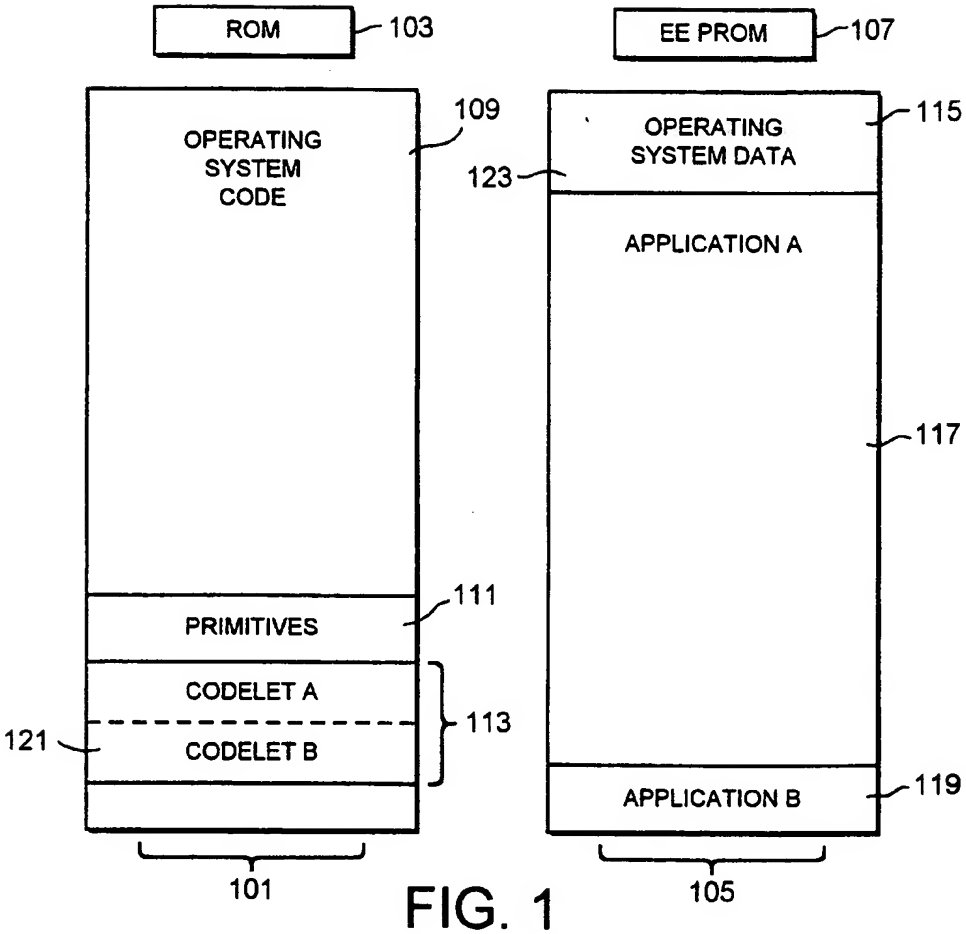

ROM
103
EE PROM
107
OPERATING SYSTEM CODE
109
OPERATING SYSTEM DATA
115
123
APPLICATION A
117
PRIMITIVES
111
CODELET A
113
121
CODELET B
APPLICATION B
119
101
105

FIG. 1

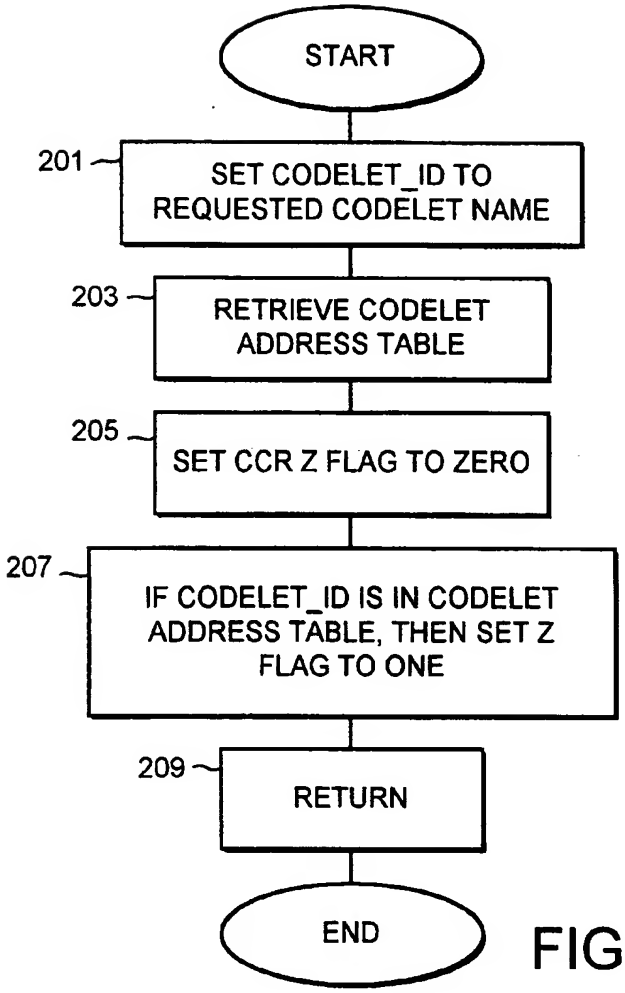

START
201
SET CODELET_ID TO REQUESTED CODELET NAME
203
RETRIEVE CODELET ADDRESS TABLE
205
SET CCR Z FLAG TO ZERO
207
IF CODELET_ID IS IN CODELET ADDRESS TABLE, THEN SET Z FLAG TO ONE
209
RETURN
END

FIG. 2

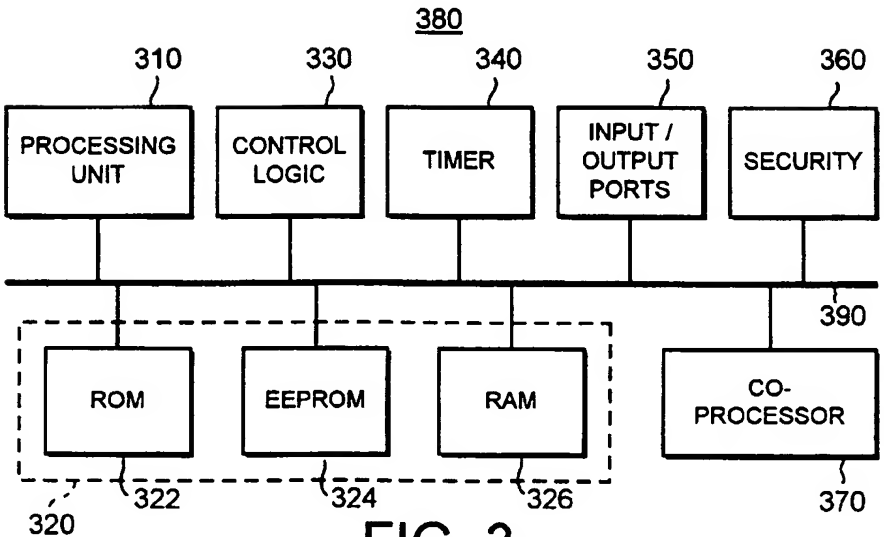

380
310
330
340
350
360
PROCESSING UNIT
CONTROL LOGIC
TIMER
INPUT / OUTPUT PORTS
SECURITY
390
ROM
EEPROM
RAM
CO-PROCESSOR
320
322
324
326
370

FIG. 3

ANNEX A TO THE DESCRIPTION

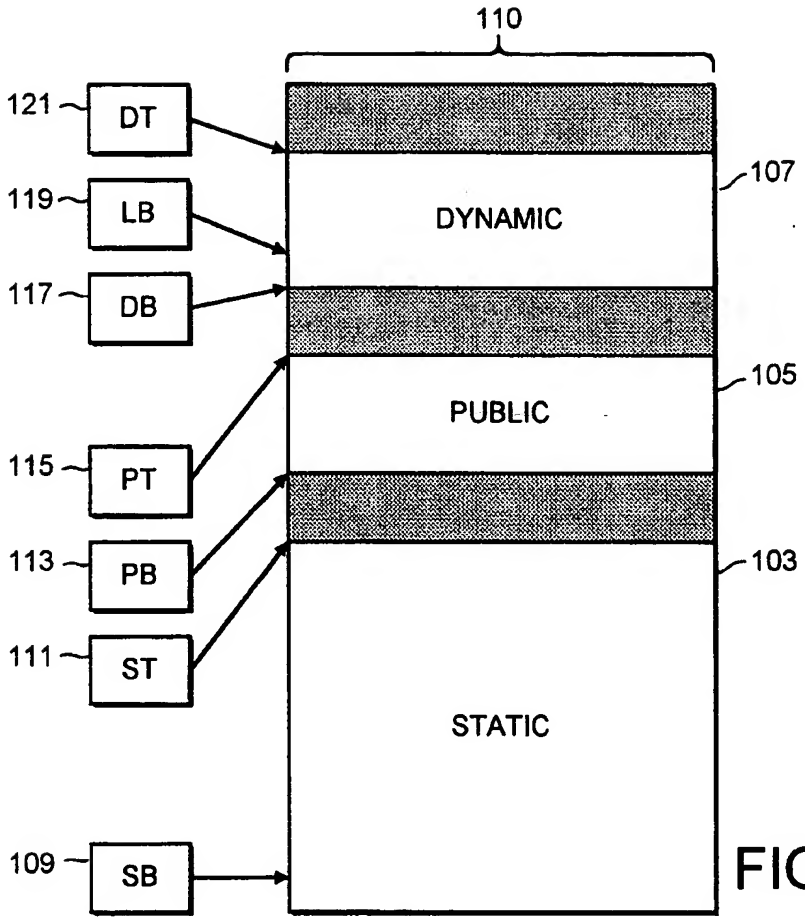


FIG. 1

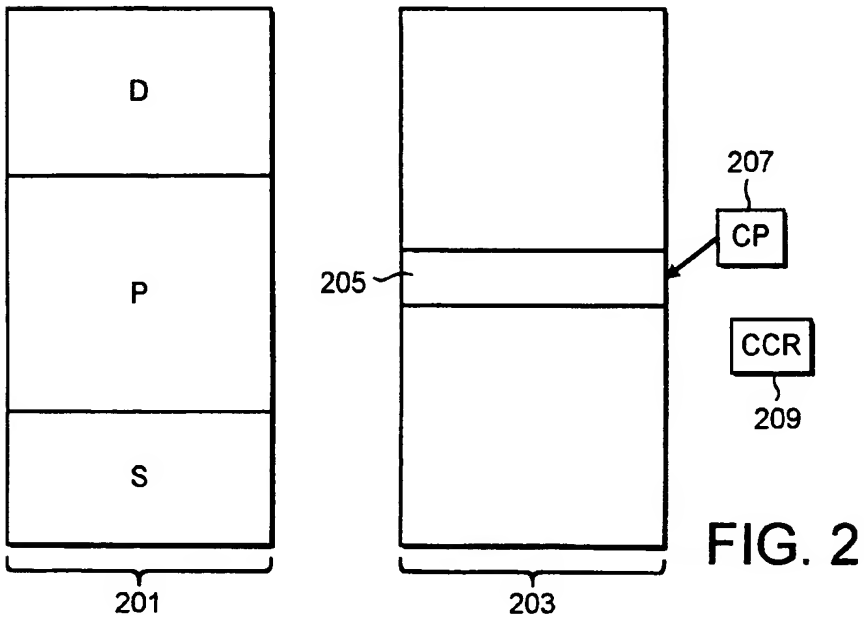


FIG. 2

ANNEX A TO THE DESCRIPTION

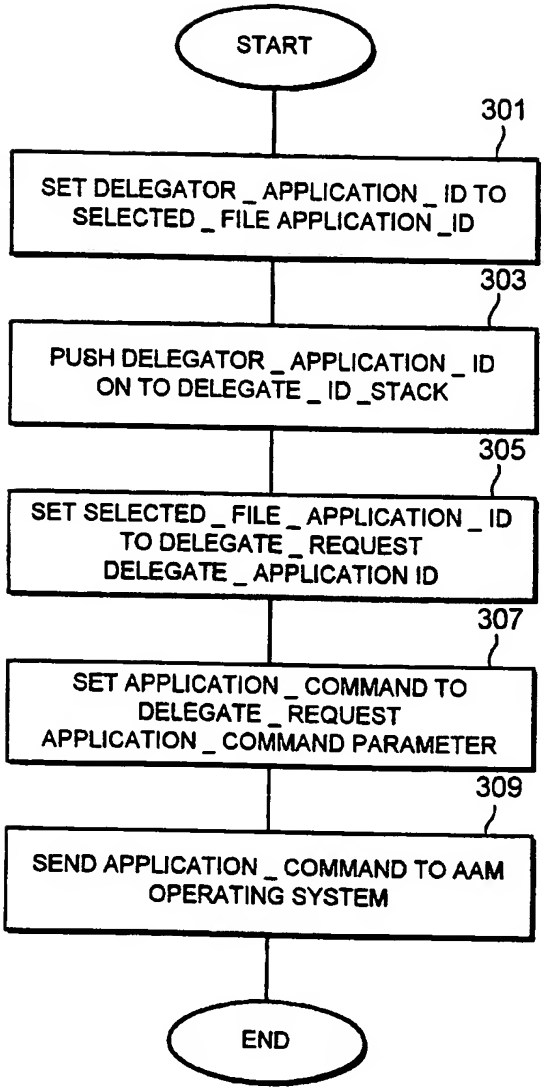


FIG. 3

ANNEX A TO THE DESCRIPTION

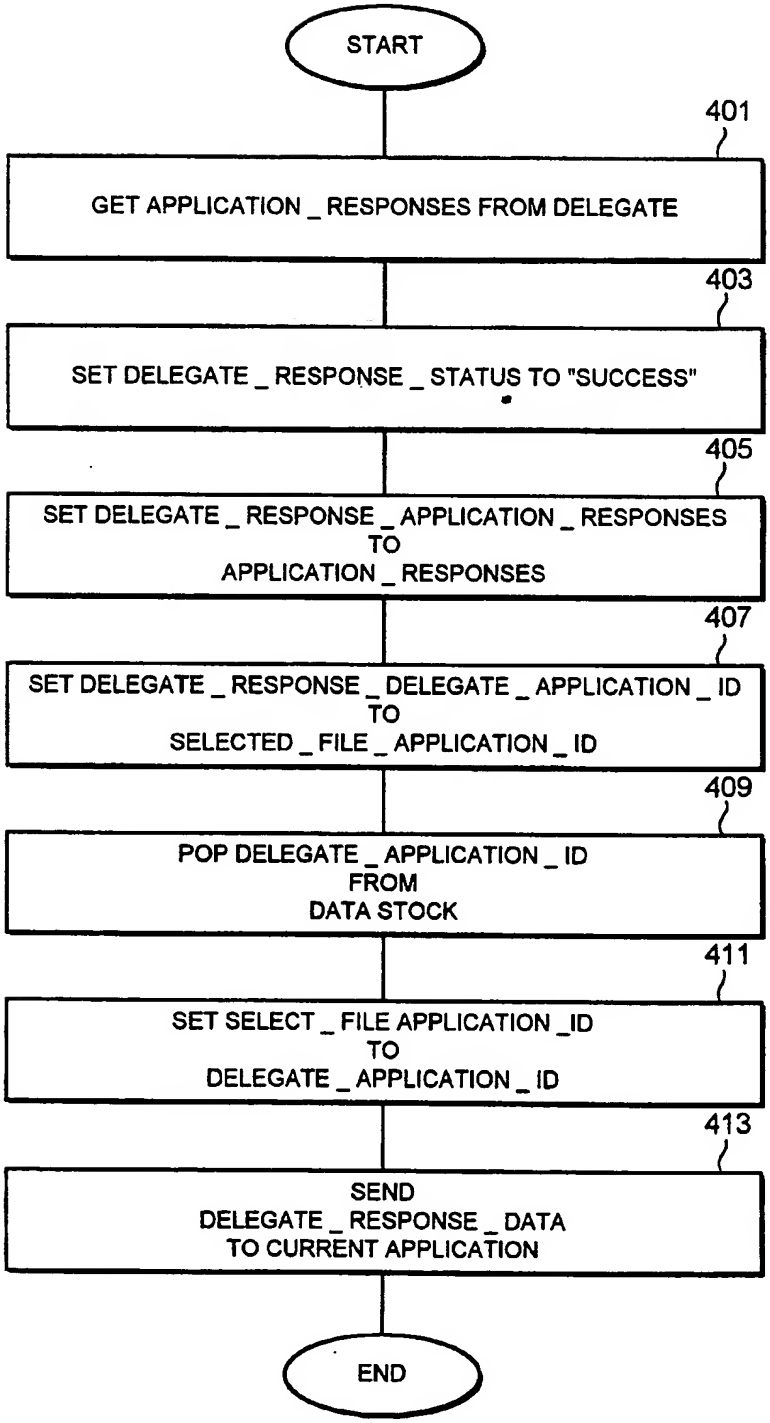


FIG. 4

ANNEX A TO THE DESCRIPTION

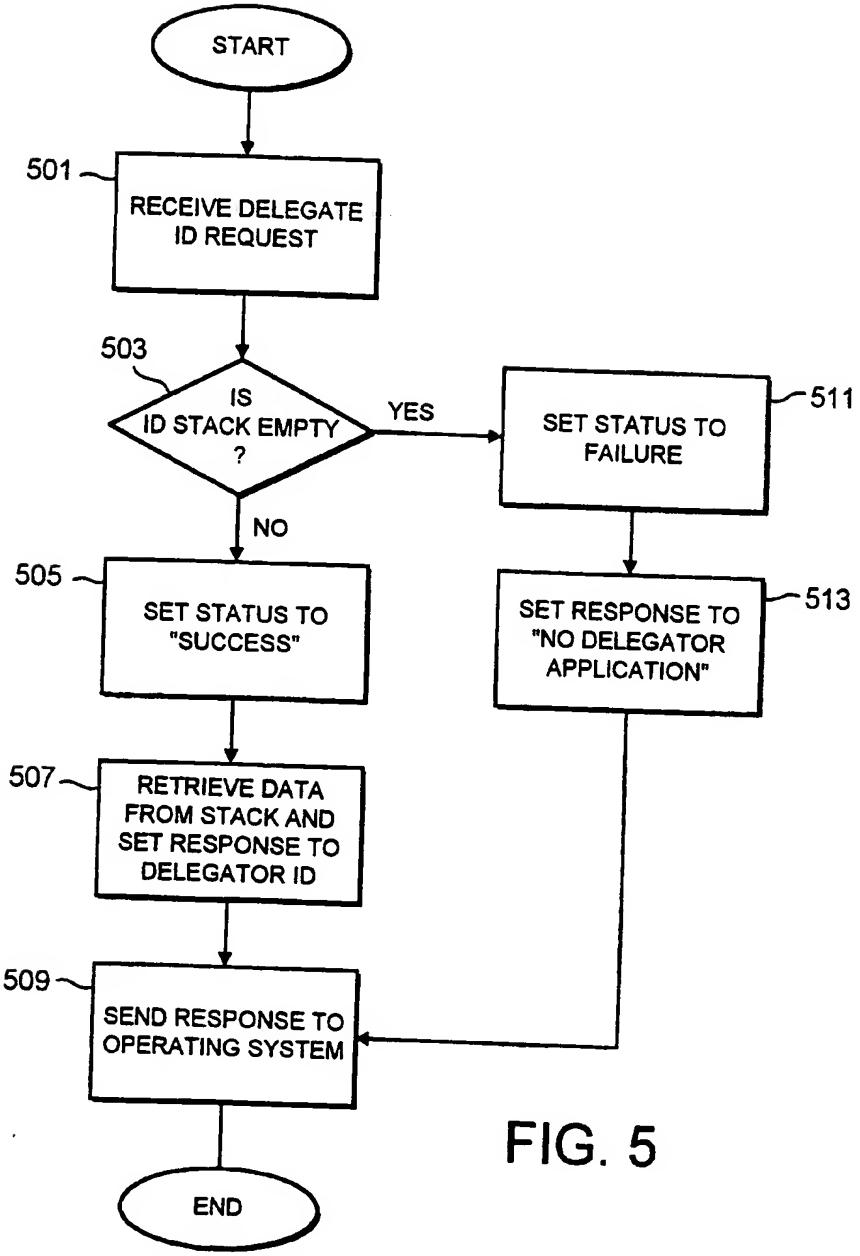


FIG. 5

ANNEX A TO THE DESCRIPTION

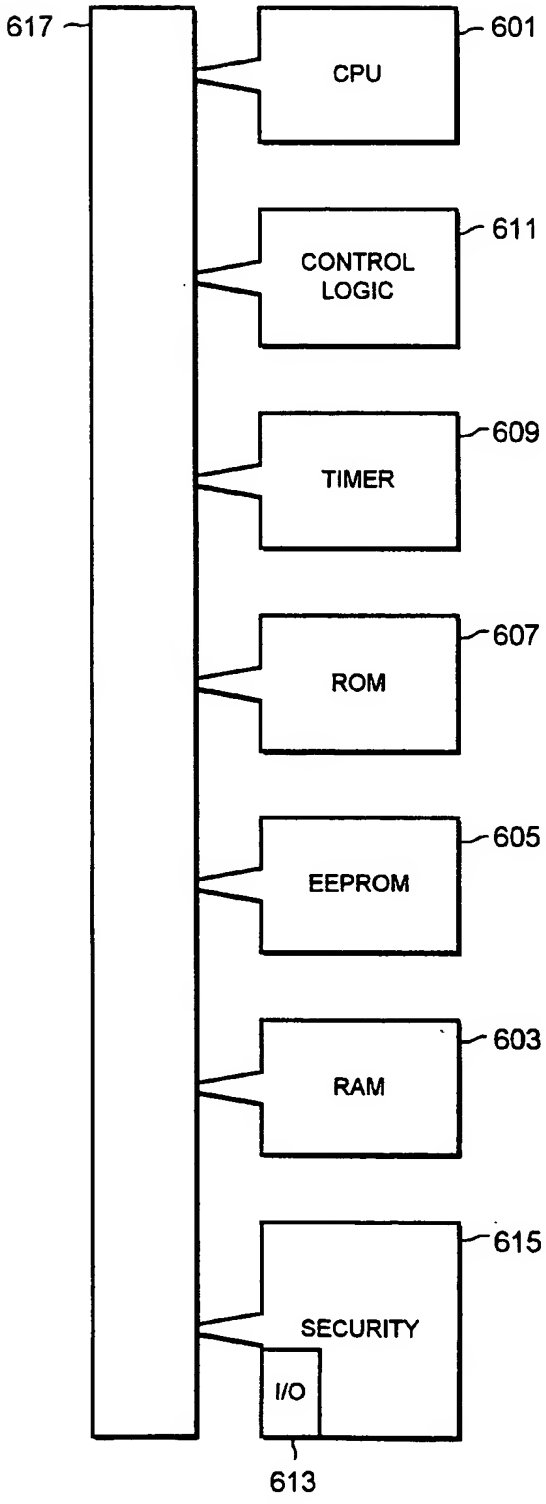


FIG. 6

ANNEX A TO THE DESCRIPTION

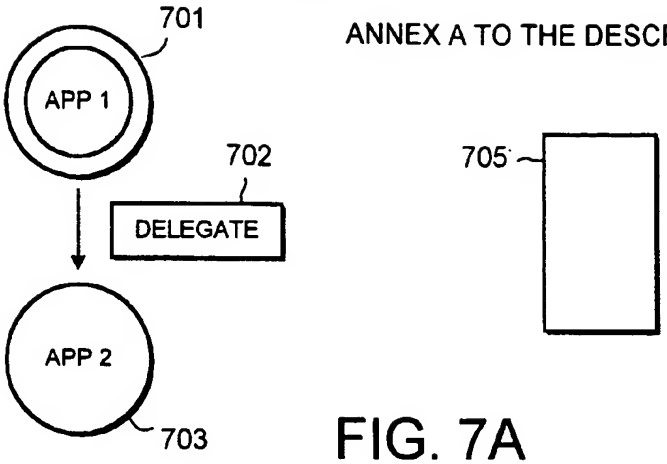


FIG. 7A

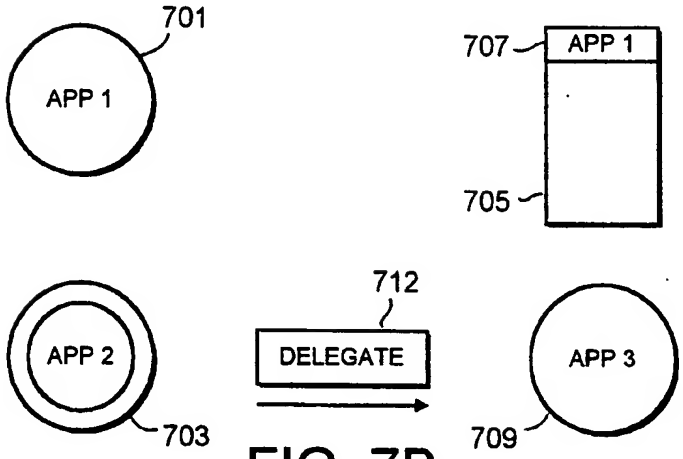


FIG. 7B

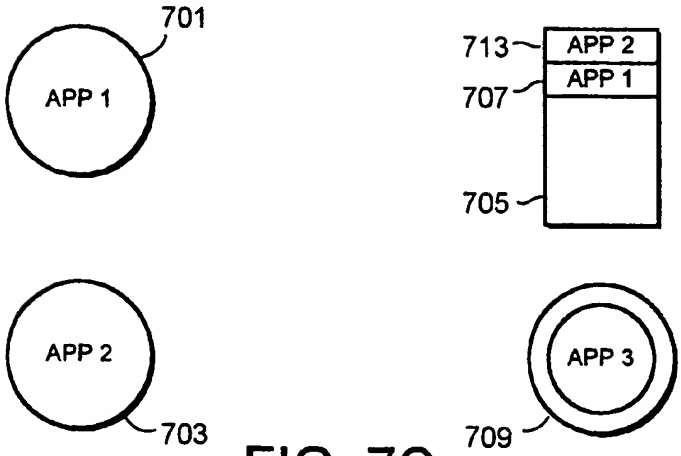


FIG. 7C

INTERNATIONAL SEARCH REPORT

Int tional Application No
PCT/GB 99/00209

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G07F7/10

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G07F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	FR 2 667 171 A (GEMPLUS CARD INTERNATIONAL) 27 March 1992 see abstract; claims; figures see page 15, line 21 - page 18, line 35	1,2,7, 12,13, 18,21, 22,24
A	EP 0 218 176 A (TOSHIBA) 15 April 1987 see abstract; claims; figures see column 3, line 42 - column 4, line 4 see column 6, line 31 - column 9, line 6	1,2,7, 12,13, 18,21, 22,24
A	US 5 682 027 A (J.M.G. BERTINA) 28 October 1997	

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
23 June 1999	30/06/1999

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	David, J

1

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Int. Jonal Application No
PCT/GB 99/00209

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	WO 96 38825 A (SYSECA) 5 December 1996	
A	EP 0 466 969 A (SIEMENS NIXFORF INFORMATIONSSYSTEME) 22 January 1992	
A	EP 0 540 095 A (PHILIPS COMPOSANTS) 5 May 1993	

1

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

Inte .onal Application No
PCT/GB 99/00209

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
FR 2667171 A	27-03-1992	NONE	
EP 0218176 A	15-04-1987	JP 62269289 A	21-11-1987
		JP 2537200 B	25-09-1996
		JP 63006690 A	12-01-1988
		JP 62082489 A	15-04-1987
		DE 3682476 A	19-12-1991
		US 4827512 A	02-05-1989
US 5682027 A	28-10-1997	AU 687760 B	05-03-1998
		AU 5332194 A	24-05-1994
		WO 9410657 A	11-05-1994
		CA 2147824 A	11-05-1994
		EP 0706692 A	17-04-1996
		NO 951575 A	26-06-1995
WO 9638825 A	05-12-1996	FR 2734934 A	06-12-1996
EP 0466969 A	22-01-1992	AT 100229 T	15-01-1994
		DE 59004248 D	24-02-1994
		US 5293577 A	08-03-1994
EP 0540095 A	05-05-1993	FR 2683357 A	07-05-1993
		DE 69223920 D	12-02-1998
		DE 69223920 T	18-06-1998
		JP 5217035 A	27-08-1993
		US 5452431 A	19-09-1995

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.